Grammar Convergence

Ralf Lämmel and Vadim Zaytsev
Software Languages Team
Universität Koblenz-Landau
What is grammar convergence?

Think of scattered grammar knowledge (say, in language documentation, parsers, object models, etc.) *how to establish relationships between the grammars, how to verify that these relationships are preserved?*
What is grammar convergence?

★ Grammar *format* to abstract from idiosyncrasies
★ Grammar *extraction* to feed into the format
★ Grammar *comparison* for spotting grammar deviations
★ Grammar *transformation*:
✦ Refactoring
✦ Extension / restriction
✦ Revision
How grammar convergence works

- Compare grammars for structural equality
- [equal] -> Pick difference
- [not equal] -> Pick grammar
  - [language-preserving] -> Refactor grammar
  - [language-revising] -> Lengthen grammar
  - [...]-increasing -> Shorten grammar
  - [...]-decreasing -> Edit grammar

Grammars

Grammar artifacts
BGF: BNF–like Grammar Format

★ BNF: symbols, composition
★ EBNF: *, +, ?
★ Production labels
★ Expression selectors
★ Universal type
★ Namespaces
g( [], [ p( [], program, +n(function)), p( [], function, (n('ID'), +n('ID'), t(=), n(expr), +n('NEWLINE')))), p( [], expr, (n(binary); n(apply); n(ifThenElse)) ), p( [], binary, (n(atom), *(n(ops), n(atom))))), p( [], apply, (n('ID'), +n(atom))), p( [], ifThenElse, (t(if), n(expr), t(then), n(expr), t(else), n(expr))), p( [], atom, (n('ID'); n('INT'); t('(', n(expr), t(')'))), p( [], ops, (t(==); t(+); t(-))) ])}
Grammar extract: XSD

g( ['Program', 'Fragment'], [ p([], 'Program', +s(function, n('Function'))), p([], 'Fragment', n('Expr')), p([], 'Function', (s(name, v(string)), +s(arg, v(string)), s(rhs, n('Expr')))), p([], 'Expr', (n('Literal');n('Argument');n('Binary');n('IfThenElse');n('Apply'))), p([], 'Literal', s(info, v(int))), p([], 'Argument', s(name, v(string))), p([], 'Binary', (s(ops, n('Ops')), s(left, n('Expr')), s(right, n('Expr')))), p([], 'Ops', (s('Equal', true);s('Plus', true);s('Minus', true))), p([], 'IfThenElse', (s(ifExpr, n('Expr')), s(thenExpr, n('Expr')), s(elseExpr, n('Expr')))), p([], 'Apply', (s(name, v(string)), +s(arg, n('Expr')))) ])

13 февраля 2009 г.
Grammar extraction

★ Get out of a source format
  ✦ Can be ANTLR, SDF, Java, XSD, HTML
★ Abstract from idiosyncrasies
  ✦ XML–isms, semantic actions, etc
★ Extraction is a generic, partial operation.
context-free syntax

Function+ -> Program
Name Name+ "=" Expr Newline+ -> Function
Expr Ops Expr -> Expr \{left,prefer,cons(binary)\}
Name Expr+ -> Expr \{avoid,cons(apply)\}
"if" Expr "then" Expr "else" Expr -> Expr \{cons(ifThenElse)\}
"(" Expr ")" -> Expr \{bracket\}
Name -> Expr \{cons(argument)\}
Int -> Expr \{cons(literal)\}

"-" -> Ops \{cons(minus)\}
"+" -> Ops \{cons(plus)\}
"==" -> Ops \{cons(equal)\}
An extractor for SDF

★ SDF basics:
☆ SDF = Syntax Def. Formalism
☆ SDF has S–G–LR as semantics.
☆ Computations over SDF:
   ☆ ASF
   ☆ Stratego
   ☆ ...
★ Extractor option:
☆ Use SDF of SDF.
☆ Use ASF over it.
☆ Construct BGF via XML.

[transform-a-production]
&C1 := sort2chardata(&N1),
&E2 := trafoSymbols(&Ss1)

=========================
trafoProd (&Ss1 -> &N1 &As1 ) =

<bf:production>
  <nonterminal>&C1</nonterminal>
  &E2
</bf:production>

[transform-empty-definition-of-nonterminal]
trafoSymbols() =
<bf:expression>
  <epsilon/>
</bf:expression>

[transform-a-definition-that-is-not-a-sequence]
trafoSymbols(&S1) = trafoSymbol(&S1)

[transform-a-definition-that-is-a-nontrivial-sequence]
&S1 &S2 &S*3 := &Ss1,
&C*1 := mapTrafoSymbol(&Ss1)

=========================
trafoSymbols(&Ss1) =
<bf:expression>
  <sequence>
    &C*1
  </sequence>
</bf:expression>
Available extractors

✓ ANTLR
✓ SDF
✓ DCG
✓ Java object models
✓ XML Schemas
✓ Language specifications
✓ …
Applying grammar convergence to the Java Language Specification

## Basic properties of the JLS sources

### Grammar class | Iteration style
---|---
`app1` | LALR(1) | left-recursive
`doc1` | none | left-recursive
`app2` | unclear | EBNF
`doc2` | none | left-recursive
`app3` | “nearly” LL(k) | EBNF
`doc3` | none | left-recursive

<table>
<thead>
<tr>
<th>Productions</th>
<th>Nonterminals</th>
<th>Tops</th>
<th>Bottoms</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app1</code></td>
<td>282</td>
<td>135</td>
<td>1</td>
</tr>
<tr>
<td><code>doc1</code></td>
<td>315</td>
<td>148</td>
<td>1</td>
</tr>
<tr>
<td><code>app2</code></td>
<td>185</td>
<td>80</td>
<td>6</td>
</tr>
<tr>
<td><code>doc2</code></td>
<td>346</td>
<td>151</td>
<td>1</td>
</tr>
<tr>
<td><code>app3</code></td>
<td>245</td>
<td>114</td>
<td>2</td>
</tr>
<tr>
<td><code>doc3</code></td>
<td>435</td>
<td>197</td>
<td>3</td>
</tr>
</tbody>
</table>
Grammar extraction for JLS

★ Use HTML representation (instead of PDF)
★ Many markup/well-formedness problems
★ Some syntax errors
★ Many obvious semantic errors
### JLS irregularities in extraction

<table>
<thead>
<tr>
<th></th>
<th>app1</th>
<th>app2</th>
<th>app3</th>
<th>doc1</th>
<th>doc2</th>
<th>doc3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary lexical decisions</td>
<td>2</td>
<td>109</td>
<td>60</td>
<td>1</td>
<td>90</td>
<td>161</td>
<td>423</td>
</tr>
<tr>
<td>Well-formedness violations</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>11</td>
<td>4</td>
<td>31</td>
</tr>
<tr>
<td>Indentation violations</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>Recovery rules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Match parentheses</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>- Metasymbol to terminal</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>27</td>
<td>7</td>
<td>42</td>
</tr>
<tr>
<td>- Merge adjacent symbols</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>- Split compound symbol</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>- Nonterminal to terminal</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>11</td>
<td>29</td>
</tr>
<tr>
<td>- Terminal to nonterminal</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>13</td>
<td>33</td>
</tr>
<tr>
<td>- Recover optionality</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Purge duplicate definitions</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>51</td>
</tr>
<tr>
<td>Total</td>
<td>11</td>
<td>123</td>
<td>92</td>
<td>24</td>
<td>181</td>
<td>238</td>
<td>669</td>
</tr>
</tbody>
</table>
Consolidation of basic metrics

<table>
<thead>
<tr>
<th>Productions</th>
<th>Nonterminals</th>
<th>Tops</th>
<th>Bottoms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>app1</strong></td>
<td>282</td>
<td>135</td>
<td>7</td>
</tr>
<tr>
<td><strong>doc1</strong></td>
<td>315</td>
<td>148</td>
<td>9</td>
</tr>
<tr>
<td><strong>app2</strong></td>
<td>185</td>
<td>80</td>
<td>11</td>
</tr>
<tr>
<td><strong>doc2</strong></td>
<td>346</td>
<td>151</td>
<td>11</td>
</tr>
<tr>
<td><strong>app3</strong></td>
<td>245</td>
<td>114</td>
<td>12</td>
</tr>
<tr>
<td><strong>doc3</strong></td>
<td>435</td>
<td>197</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Productions</th>
<th>Nonterminals</th>
<th>Tops</th>
<th>Bottoms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jls1</strong></td>
<td>278</td>
<td>132</td>
<td>7</td>
</tr>
<tr>
<td><strong>jls2</strong></td>
<td>182</td>
<td>75</td>
<td>7</td>
</tr>
<tr>
<td><strong>jls3</strong></td>
<td>236</td>
<td>109</td>
<td>7</td>
</tr>
<tr>
<td><strong>jls12</strong></td>
<td>182</td>
<td>75</td>
<td>7</td>
</tr>
<tr>
<td><strong>jls123</strong></td>
<td>236</td>
<td>109</td>
<td>7</td>
</tr>
<tr>
<td><strong>doc12</strong></td>
<td>347</td>
<td>152</td>
<td>7</td>
</tr>
<tr>
<td><strong>doc123</strong></td>
<td>440</td>
<td>201</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure 1. Grammar classes and iteration style for the JLS grammars.

Figure 2. Simple grammar metrics.

Figure 5. Metrics for the transformed grammars.

Figure 6: Effort measurements per target in the convergence graph for the JLS.
Grammar comparison

★ Compare grammars structurally.
★ Apply simple algebraic laws on grammars.
★ Provide suggestive input for transformation.
Grammar transformation

★ Performing post-extraction activities
★ Refactoring for structural equivalence
★ Extension to cover missing language construct
★ Restriction to abstract away “irrelevant” constructs
★ Relaxation to abstract away “irrelevant” precision
★ Replacement to fix accidental deviations
★ Capture and document language differences
A fragment of concrete syntax. What if we want to derive the abstract syntax?

expr: ...;
atom: ID | INT | '(' expr ')';

Need to project away "(" & ")"

Need to merge "expr" & "atom"

Alternative needs to go entirely
A transformation sequence

expr : ...;
atom : ID | INT | '(' expr ')';

abstractize
expr : ...;
atom : ID | INT | expr;

vertical
expr : ...;
atom : ID;
atom : INT;
atom : expr;

unite
expr : ...;
expr : ID;
expr : INT;
expr : expr;

abridge
XBGF Operator Suite

- Semantics-preserving (refactoring)
  - rename, introduce, eliminate
  - fold, unfold, extract, inline
  - factor, distribute, horizontal, vertical
  - yaccify, deyaccify, massage
  - designate, unlabel
  - ...

13 февраля 2009 г.
XBGF Operator Suite

★ Semantics–increasing/–decreasing
✦ appear, disappear
✦ narrow, widen
✦ add, remove
✦ upgrade, downgrade
✦ unite
✦ ...

февраля 2009 г.
Semantics–revising
- undefine, define, redefine
- inject, project, permute
- abstractize, concretize
- replace
A more detailed convergence tree

ANTLR -> DCG

SDF -> XSD

OM -> JAXB

Concrete

- Remove Layers
- Define Lex
- Strip Terminals
- Permute Args

Abstract

- Reroot
- MKSignature

Limit

Java

Trim XSD -> Trim OM -> Trim JAXB

Rename SDF -> Rename OM -> Rename JAXB

Permute Args

Unerase
Transformation statistics for JLS

<table>
<thead>
<tr>
<th></th>
<th>jls1</th>
<th>jls2</th>
<th>jls3</th>
<th>jls12</th>
<th>jls123</th>
<th>doc12</th>
<th>doc123</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of lines</td>
<td>600</td>
<td>4807</td>
<td>9469</td>
<td>4285</td>
<td>2934</td>
<td>1491</td>
<td>3072</td>
<td>26658</td>
</tr>
</tbody>
</table>
| Number of
| transformations | 62   | 367  | 538  | 287   | 120    | 70    | 133    | 1577   |
| semantics-preserving | 40   | 278  | 398  | 235   | 87     | 25    | 73     | 1136   |
| semantics-increasing or -decreasing | 22   | 78   | 127  | 50    | 32     | 38    | 56     | 403    |
| semantics-revising | —    | 11   | 13   | 2     | 1      | 7     | 4      | 38     |
| Number of issues | 8    | 38   | 47   | 25    | 17     | 32    | 40     | 207    |
| recoveries      | —    | 7    | 8    | —     | —      | 7     | 4      | 26     |
| corrections      | 5    | 22   | 22   | 2     | —      | 10    | 7      | 68     |
| extensions       | —    | —    | —    | 17    | 14     | 15    | 28     | 74     |
| optimizations    | 3    | 9    | 17   | 6     | 3      | —     | 1      | 39     |
Conclusion and future work

- Synchronise scattered grammar knowledge
- Further consolidation of operator suite
- Co-transformation of parse-trees possible
- Semi-automatic approach desirable
- Additional techniques for priorities
- Alignment with metamodeling-based work
Thank you!

★ Questions?
★ Comments?
★ Software Language Processing Suite is here: http://sourceforge.net/projects/slps/